# Unit – 1

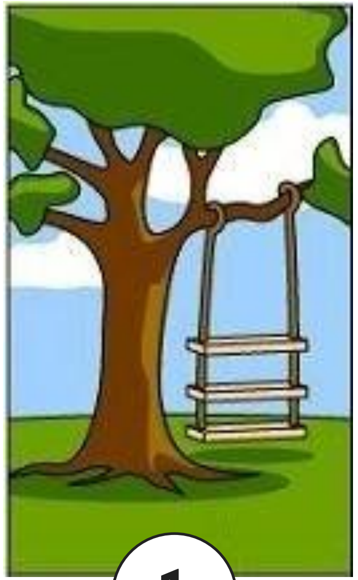
# Introduction to Software and Software Engineering

## ✓ Outline

- Software, Characteristics of Software, Software Application Domains
- Software Engineering, Software Engineering Layered Approach
- Software Process, Process Framework Activities , Umbrella Activities
- Software Myths
  - Management Myth
  - Customer Myth
  - Practitioner's/Developer Myth)
- Software Process Models
  - The Waterfall Model
  - Incremental Process Model
  - Prototyping Model, Spiral Model
  - Spiral Model
  - Rapid Application Development Model (RAD)
- Component based Development

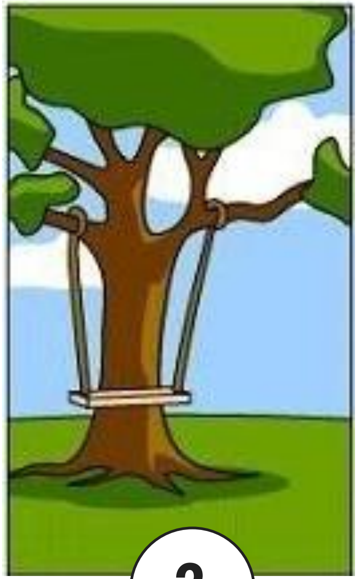
# Why to Study Software Engineering?

## Software Development Life Cycle without Software Engineering



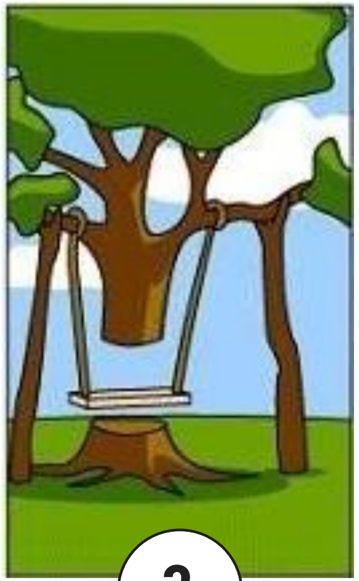
1

How the Customer Explains Requirement



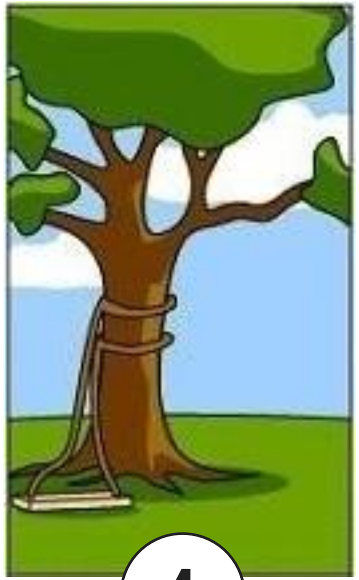
2

How the Project Leader understand it



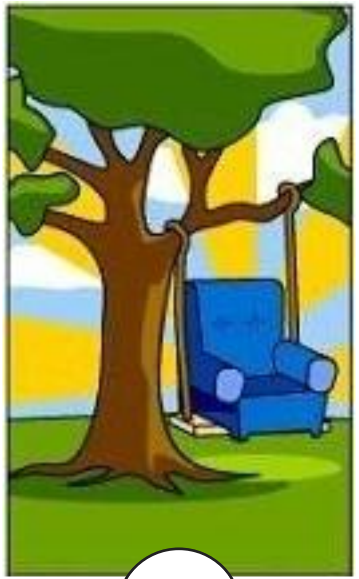
3

How the System Analyst design it



4

How the Programmer Works on it

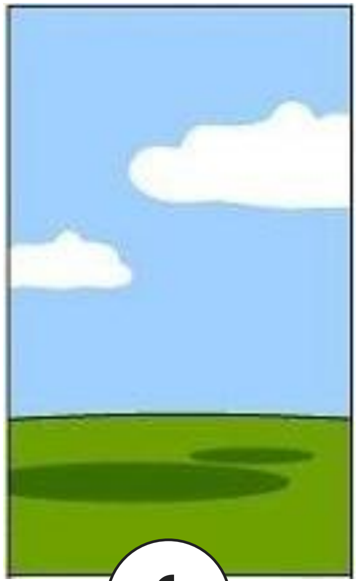


5

How the Business Consultant describe it

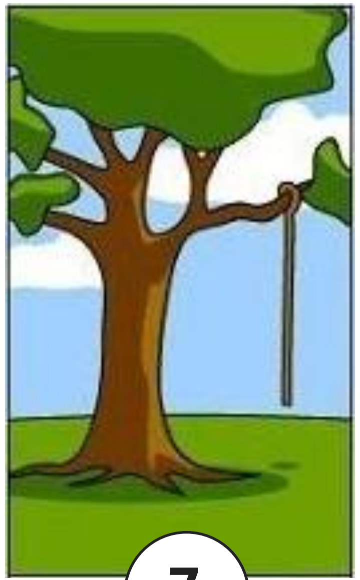
# Why to Study Software Engineering?

## Software Development Life Cycle without Software Engineering



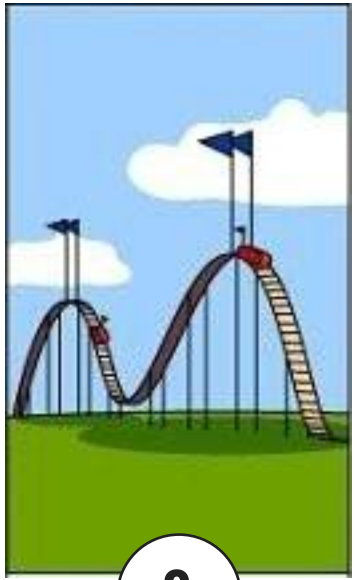
6

How the Project documented



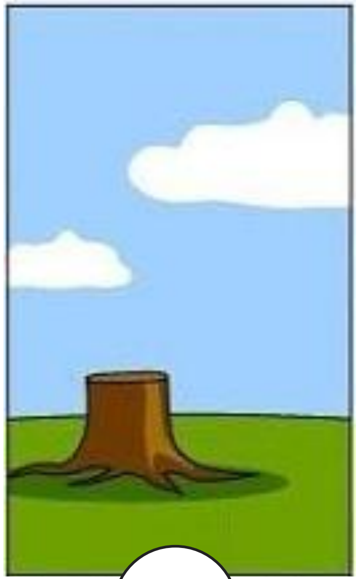
7

What Operations Installed



8

How the Customer billed



9

How it was supported

10

What the customer really needed

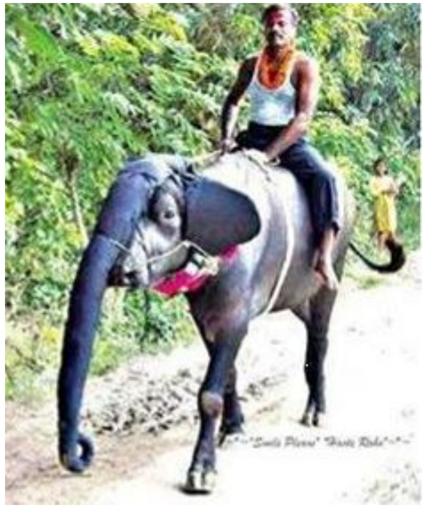
# SDLC **without** Software Engineering

## Customer Requirement

- Have one trunk
- Have four legs
- Should carry load both passenger & cargo
- Black in color
- Should be herbivorous

## Solution

- Have one trunk
- Have four legs
- Should carry load both passenger & cargo
- Black in color
- Should be herbivorous



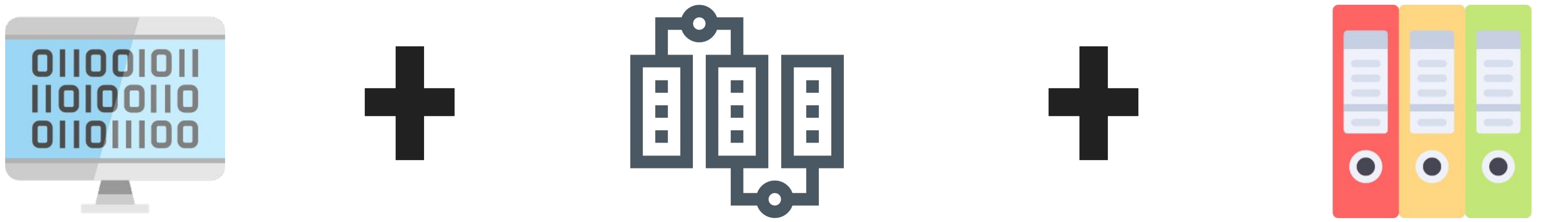
Our value added, also gives milk

The software development **process** needs to be **engineered** to avoid the **communication gap** & to **meet the actual requirements** of customer within **stipulated budget** & **time**

# What is Software?

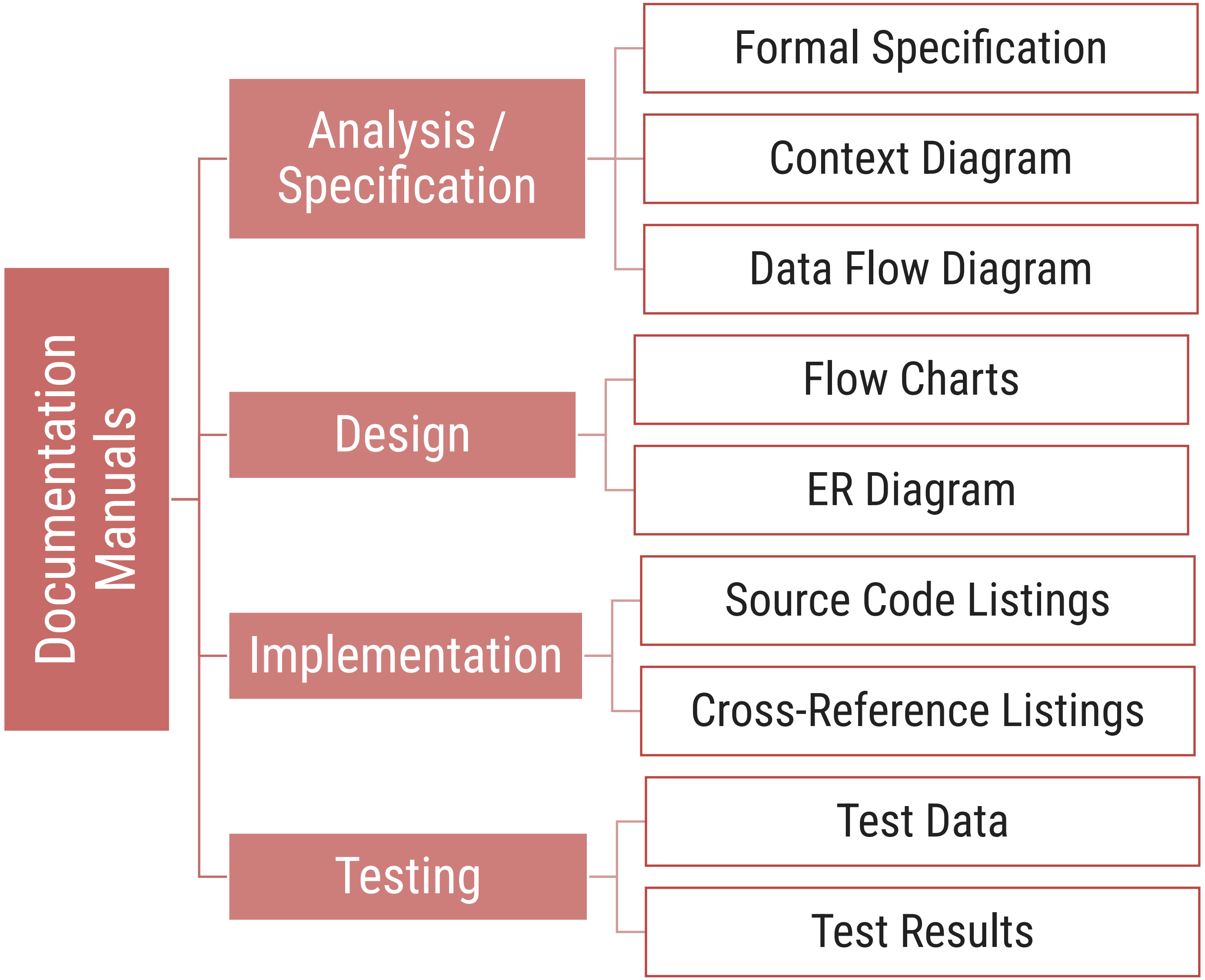
## Software is

1) **Computer program** that when executed provide desired features, function & performance
2) **Data Structure** that enable programs to easily manipulate information
3) **Descriptive information** in both hard and soft copy that describes the operation and use of programs

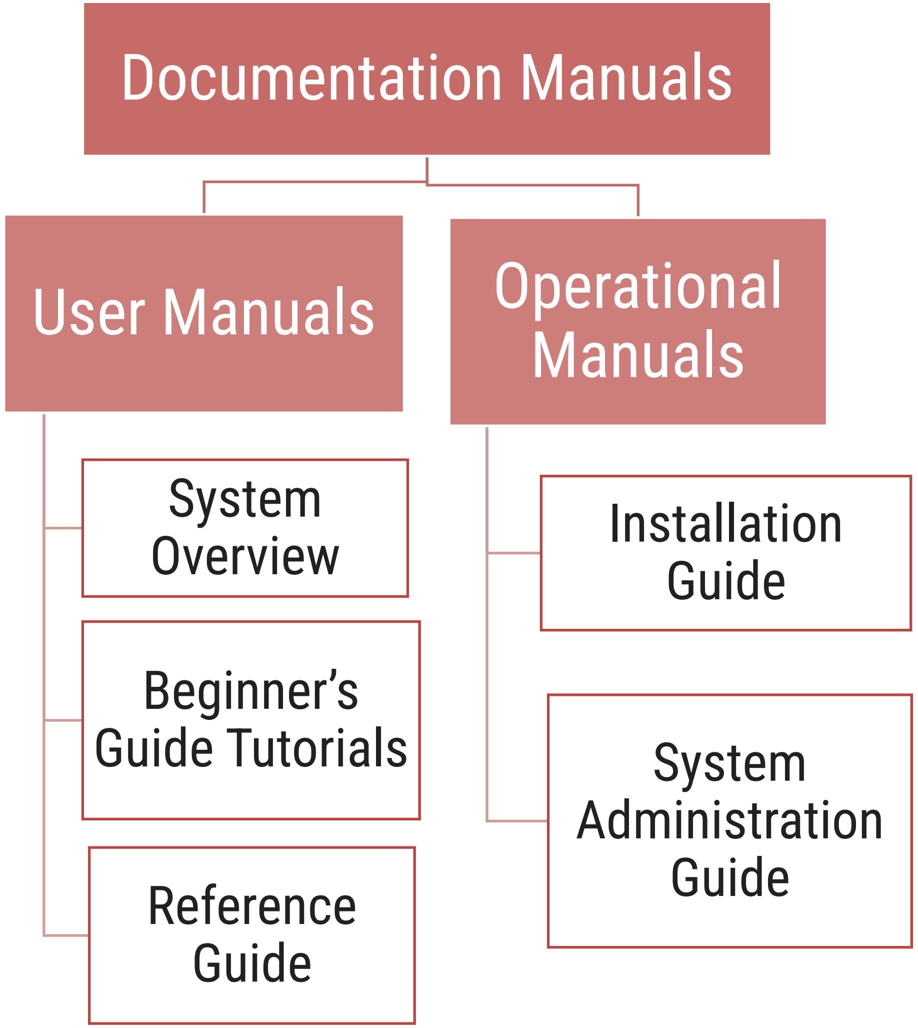


**Computer Program** + **Data Structure** + **Documents Soft & Hard**

# List of documentation & manuals



- Documentation Manuals
  - Analysis / Specification
    - Formal Specification
    - Context Diagram
    - Data Flow Diagram
  - Design
    - Flow Charts
    - ER Diagram
  - Implementation
    - Source Code Listings
    - Cross-Reference Listings
  - Testing
    - Test Data
    - Test Results



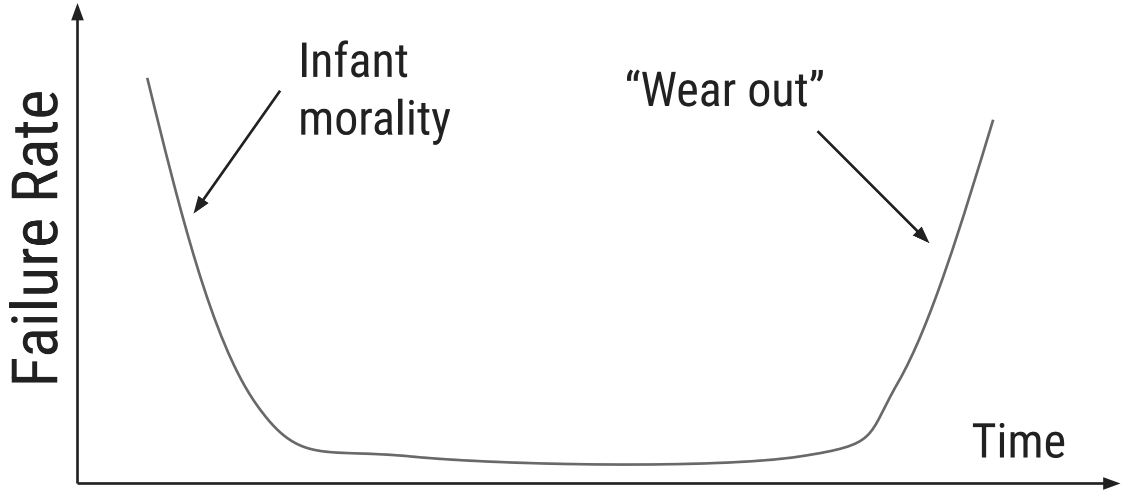
- Documentation Manuals
  - User Manuals
    - System Overview
    - Beginner's Guide Tutorials
    - Reference Guide
  - Operational Manuals
    - Installation Guide
    - System Administration Guide

# Characteristics of Software

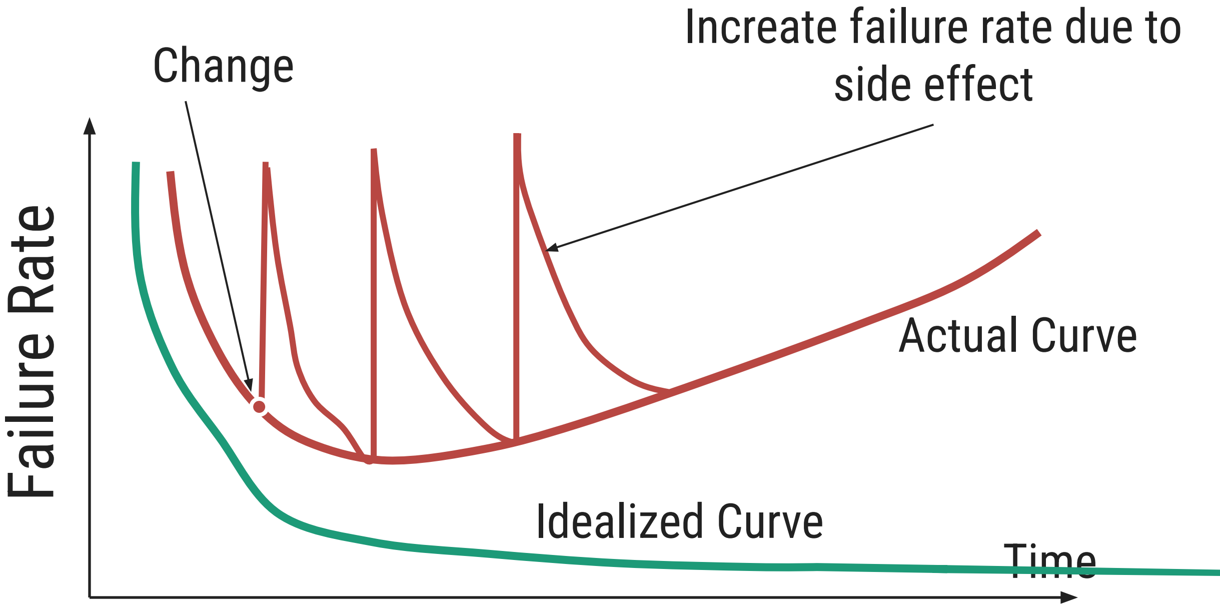
## Software is developed or engineered

- It is not manufactured like hardware
  - Manufacturing phase can introduce quality problem that are nonexistent (or easily corrected) for software
  - Both requires construction of “product” but approaches are different

## Software doesn’t “wear-out”



Bathtub curve of hardware failure



Software failure curve

# Software Engineering

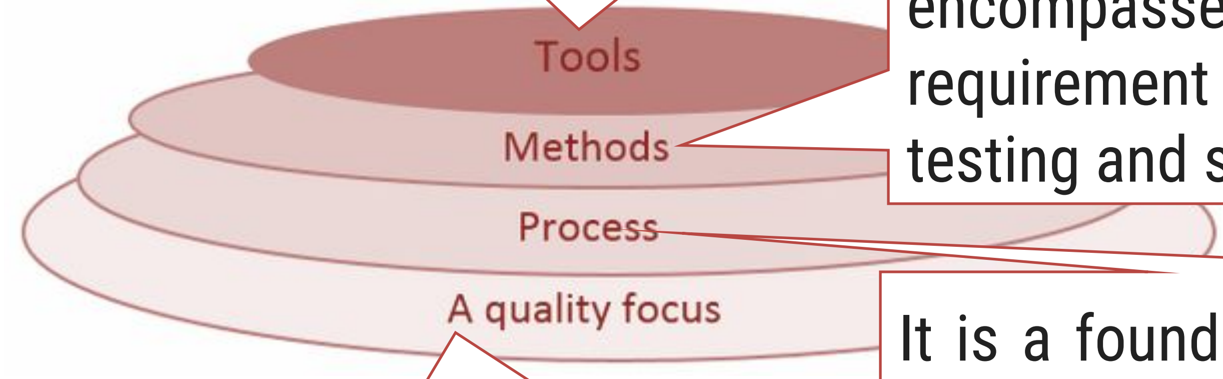
**Software engineering** is the establishment and use of **sound engineering principles** in order to obtain **economically software** that is **reliable and works** efficiently in **real machines**.

Software Engineering is the science and art of building (designing and writing programs) a software systems that are:

1) on **time**
2) on **budget**
3) with acceptable **performance**
4) with **correct operation**

# Software Engineering Layered Approach

Software Engineering Tools **allows automation of activities** which helps to perform systematic activities. A system for the support of software development, called **computer-aided software engineering** (CASE). **Examples:** Testing Tools, Bug/Issue Tracking Tools etc…



It provides **technical how-to's** for building software, it encompasses **many tasks** including communication, requirement analysis, design modeling, program construction, testing and support

It is a foundation of Software Engineering, It is the **glue** the holds the **technology layers**, It **defines** a **framework activities**

Defines continuous **process improvement principles**

# Software Engineering Layered Approach Cont.

## Quality

- Main principle of Software Engineering is Quality Focus.
- An **engineering approach** must have a **focus on quality**.
- Total Quality Management **(TQM)**, **Six Sigma**, **ISO** 9001, ISO 9000-3, CAPABILITY MATURITY MODEL **(CMM)**, **CMMI** & similar approaches encourages a continuous process improvement culture

## Process Layer

- It is a foundation of Software Engineering, It is the glue the holds the technology layers together and enables logical and timely development of computer software.
- It **defines** a **framework** with activities for effective delivery of software engineering technology
- It establish the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.

# Software Engineering Layered Approach Cont.

## Method

- It provides **technical how-to's** for building software
- It **encompasses many tasks** including communication, requirement analysis, design modeling, program construction, testing and support

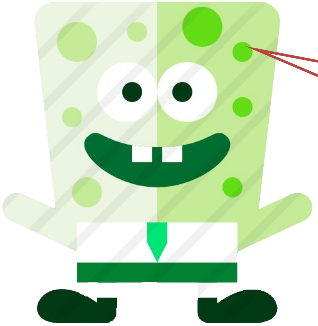
## Tools

- Software engineering tools provide automated or semiautomated support for the process and the methods
- Computer-aided software engineering (**CASE**) is the scientific application of a **set of tools** and **methods** to a software system which is meant to **result in high-quality, defect-free, and maintainable software products**.
- CASE tools automate many of the activities involved in various life cycle phases.

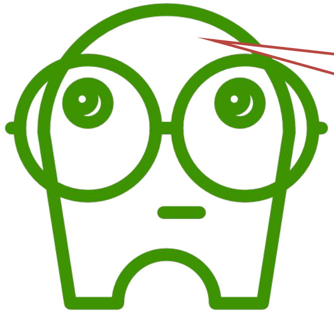
# Software Myths

Beliefs about software and the process used to build it.

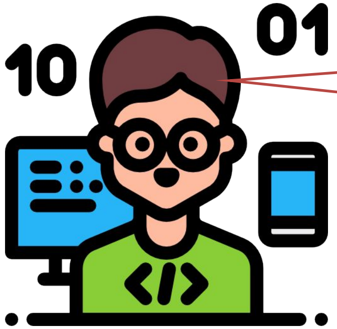
"**Misleading Attitudes that cause serious problem**" are myths.



Management Myths



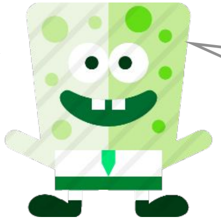
Customer Myths



Practitioner's (Developer) Myths

# Management myth - 1 & 2

We **have standards and procedures** to build a system, which is enough.



We have **the newest computers and development tools**.

## Reality

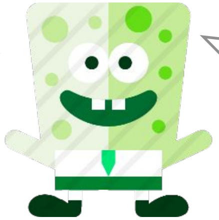
- Are software **practitioners aware** of standard’s existence?
- Does it **reflect modern software engineering** practice?
- Is it **complete**?
- Is it streamlined to **improve time to delivery** while **still maintaining a focus on quality**?
- In many cases, the answer to all of these questions is "no.“

## Reality

- It **takes much more than the latest model** computers to do high-quality software development.
- **Computer-aided software engineering (CASE)** tools are more important than hardware.

# Management myth - 3 & 4

We **can add more programmers** and can catch up the schedule.



I **outsourced the development** activity, now I **can relax** and **can wait** for the **final** working **product**.

## Reality

- Software **development is not a mechanistic process** like manufacturing.
- In the words of Fred Brooks : "**adding people to a late software project makes it later.**"
- **People** who were **working** must **spend time educating** the newcomers.
- People can be added but only **in a planned and well-coordinated** manner.

## Reality

- If an **organization** does **not understand how to manage** and **control** software projects internally, it will invariably struggle when it outsources software projects.

# Customer myth - 1 & 2

A **general statement of objectives** (requirements) is **sufficient** to start a development.

**Requirement Changes** can be **easily accommodated** because software is very flexible.

## Reality

- Comprehensive (**detailed**) **statements** of requirements is not always possible, an **ambiguous** (unclear) **"statement of objectives"** can lead to disaster.
- Unambiguous (clear) requirements can be gathered only through effective and continuous communication between customer and developer.

## Reality

- It is true that software **requirements change**, but the **impact** of change **varies with the time** at which it is introduced.
- When requirements changes are requested early the cost impact is relatively small.

# Practitioner's (Developer) myth – 1 & 2

**Once** we **write** the **program**, our **job is done**.

## Reality

- Experts say "**the sooner you begin 'writing code', the longer it will take you to get done.**"
- Industry data indicates that 60 to 80 % effort expended on software will be after it is delivered to the customer for the first time.

I **can't** access **quality until** it is **running**.

## Reality

- One of the most effective software **quality assurance mechanisms** can be **applied from** the **beginning** of a project - **the technical review**.
- Software reviews are more effective "quality filter" than testing for finding software defects.

# Practitioner's (Developer) myth – 3 & 4

Working **program** is the **only** **deliverable** work **product**.

**Software engineering** is about **unnecessary** documentation.

## Reality

- **A working program** is only one **part of** a **software configuration**.
- A variety of work products (e.g., **models, documents, plans**) provide a foundation for successful engineering and, more important, guidance for software support.

## Reality

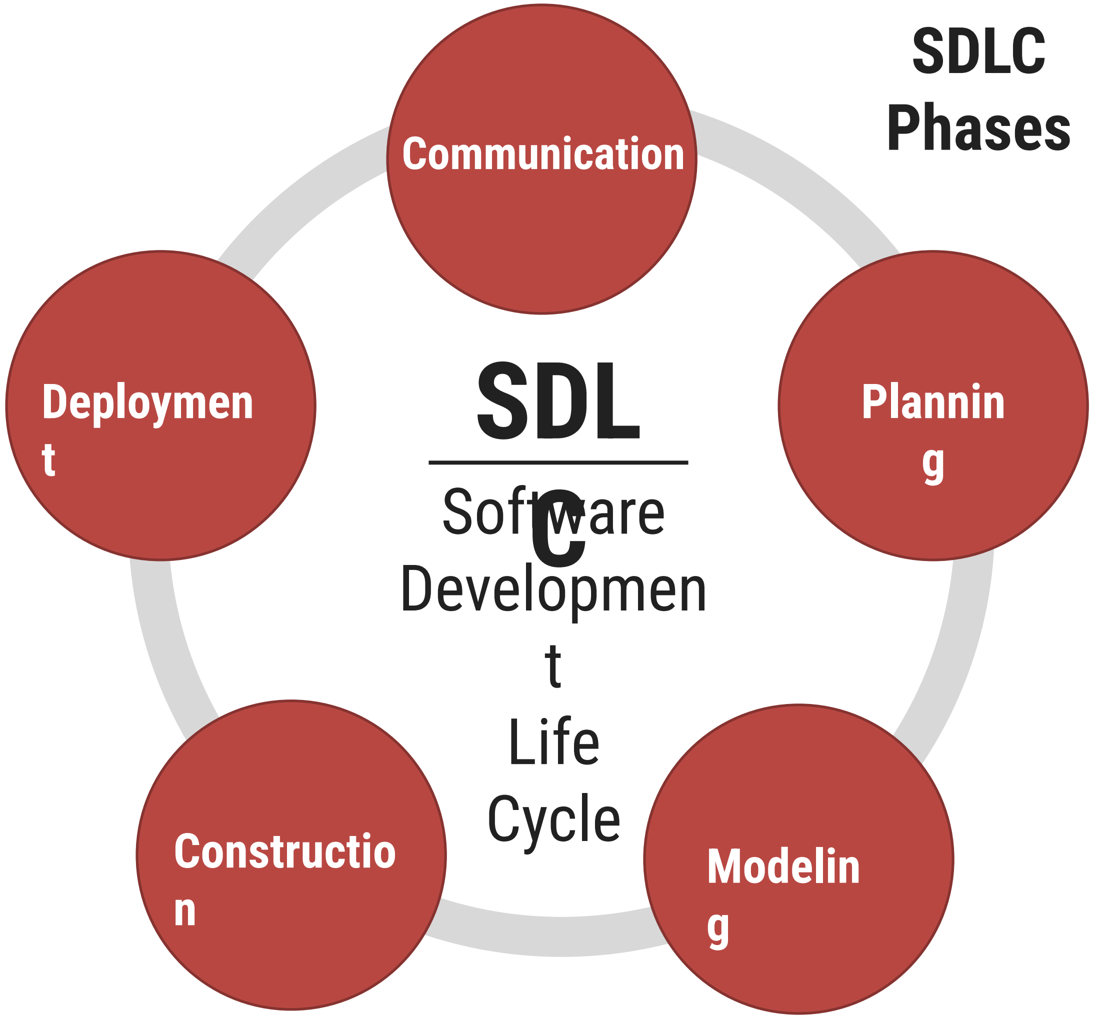
- Software engineering is not about creating documents. It is about **creating a quality product**.
- Better quality leads to reduced rework. And reduced rework results in faster delivery times.

# Software Process Models

The **process model** is the abstract representation of process.

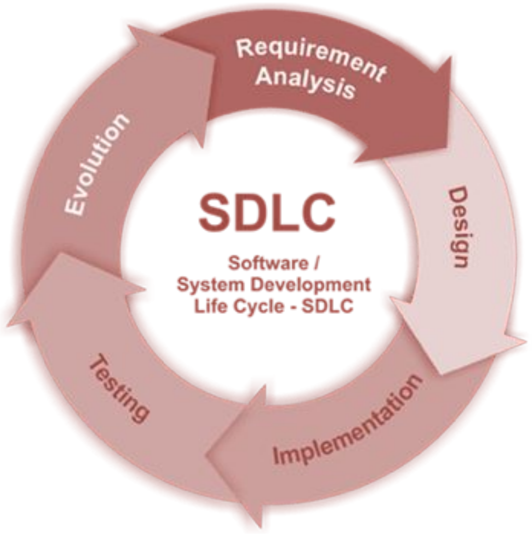
- Also known as **Software development life cycle (SDLC)** or Application development life cycle Models
- Process models **prescribe** a distinct set of **activities, actions, tasks and milestones (deliverables)** required to engineer high quality software.
- Process **models are not perfect,** but **provide roadmap** for software engineering work.
- Software models provide stability, control and organization to a process that if not managed can easily get out of control.
- Software process models are adapted (adjusted) to meet the needs of software engineers and managers for a specific project.

**SDLC Phases**



# Different Process Models

- Process model is selected based on different parameters
  - Type of the project & people
  - Complexity of the project
  - Size of team
  - Expertise of people in team
  - Working environment of team
  - Software delivery deadline

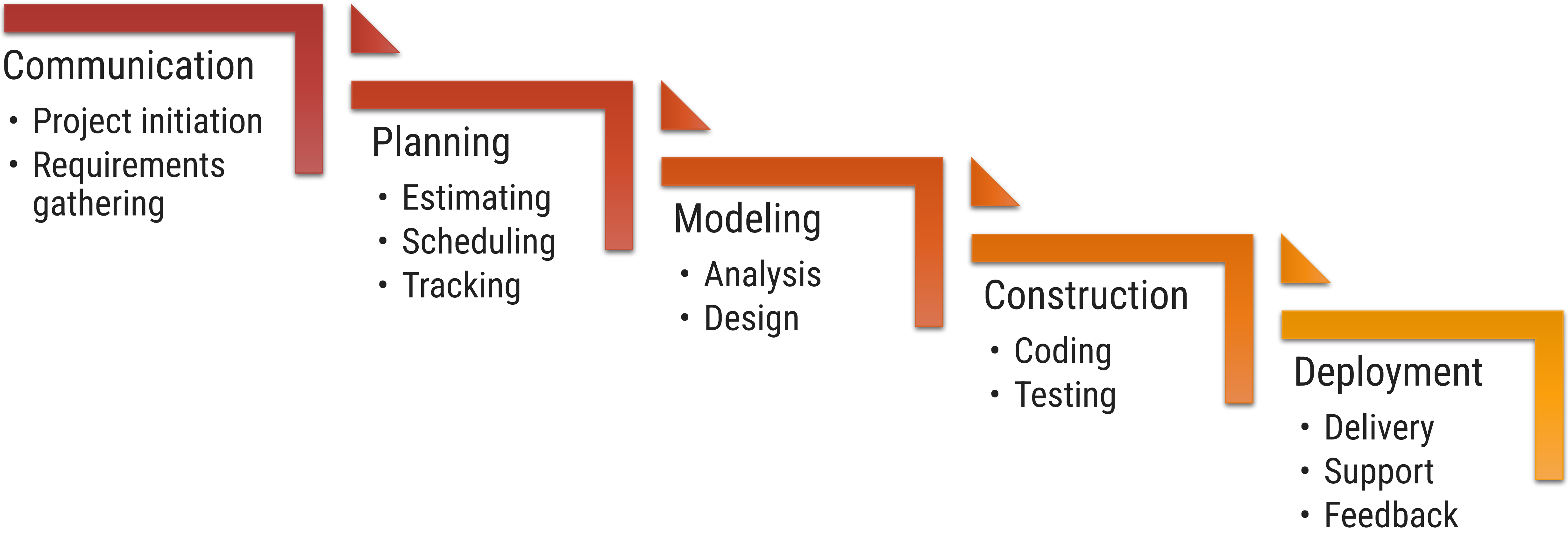


## Process Models

- Waterfall Model (Linear Sequential Model)
- Incremental Process Model
- Prototyping Model
- The Spiral Model
- Rapid Application Development Model
- Agile Model

# The Waterfall Model

**Classic life cycle** or **linear sequential model**



Communication

- Project initiation
- Requirements gathering

Planning

- Estimating
- Scheduling
- Tracking

Modeling

- Analysis
- Design

Construction

- Coding
- Testing

Deployment

- Delivery
- Support
- Feedback

When **requirements** for a problems are **well understood** then this model is used in which **work flow** from communication to deployment is **linear**

# The Waterfall Model

## When to use ?

- **Requirements** are very well **known**, **clear** and **fixed**
- Product **definition** is **stable**
- **Technology** is **understood**
- There are **no ambiguous** (unclear) **requirements**
- Ample (**sufficient**) **resources** with required **expertise** are **available** freely
- The **project** is **short**

## Advantages

- **Simple to implement** and manage

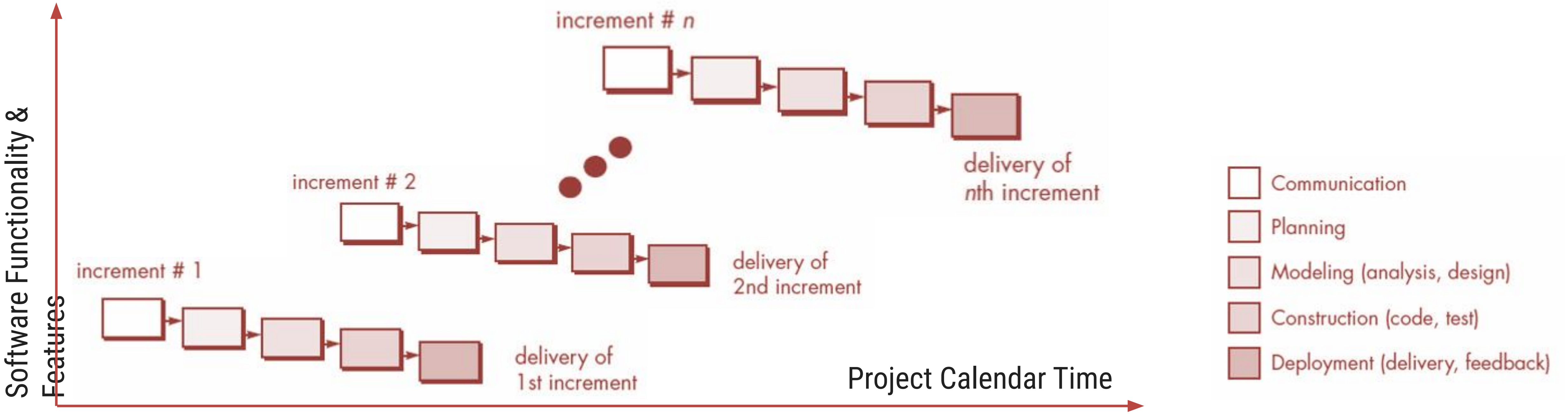
## Drawbacks

- **Unable to accommodate changes** at **later stages**, that is required in most of the cases.
- **Working version** is **not available** during development. Which can lead the development with major mistakes.
- **Deadlock can occur** due to delay in any step.
- **Not suitable** for **large projects**.

# Incremental Process Model

- There are many situations in which **initial software requirements** are reasonably **well defined**, but the **overall scope of the development** effort prevent a purely linear process.
- In addition, there may be a **compelling need** to provide a **limited set of software functionality** to users **quickly** and then **refine and expand on that functionality** in later software releases.
- In such cases, there is a need of a process model that is designed to produce the software in increments.

# Incremental Process Model


Software Functionality & Features
increment # n
increment # 2
increment # 1
delivery of nth increment
delivery of 2nd increment
delivery of 1st increment
Project Calendar Time
Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)

- The incremental model **combines** elements of **linear** and **parallel** process flows.
- This model applies linear sequence in a iterative manner.
- Initially **core working product** is **delivered**.
- **Each** linear **sequence** produces deliverable **“increments”** of the software.

# Incremental Process Model

e.g. **word-processing software** developed **using** the **incremental model**

- It might deliver basic file management, editing and document production functions in the first increment
- more sophisticated editing in the second increment;
- spelling and grammar checking in the third increment; and
- advanced page layout capability in the fourth increment.

## When to use ?

- When the **requirements** of the **complete** system are clearly **defined** and understood but **staffing is unavailable** for a **complete implementation** by the business deadline.

## Advantages

- Generates **working software quickly** and early during the software life cycle.
- It is **easier to test** and debug during a smaller iteration.
- **Customer** can **respond** to each built.
- **Lowers initial** delivery **cost**.
- **Easier** to **manage risk** because risky pieces are identified and handled during iteration.

# Evolutionary Process Models

- When a set of **core product** or system requirements is **well understood** but the **details of product** or system extensions have **yet to be defined**.
- In this situation there is **a need of process model** which specially designed to accommodate **product** that **evolve with time**.
- **Evolutionary Process Models** are specially meant for that which produce an increasingly more complete version of the software with each iteration.
- Evolutionary Models are **iterative**.
- They are characterized in a manner that enables you to develop **increasingly more complete versions of the software**.
- Evolutionary models are
  - **Prototyping Model**
  - **Spiral Model**

# Prototyping model

## When to use ?

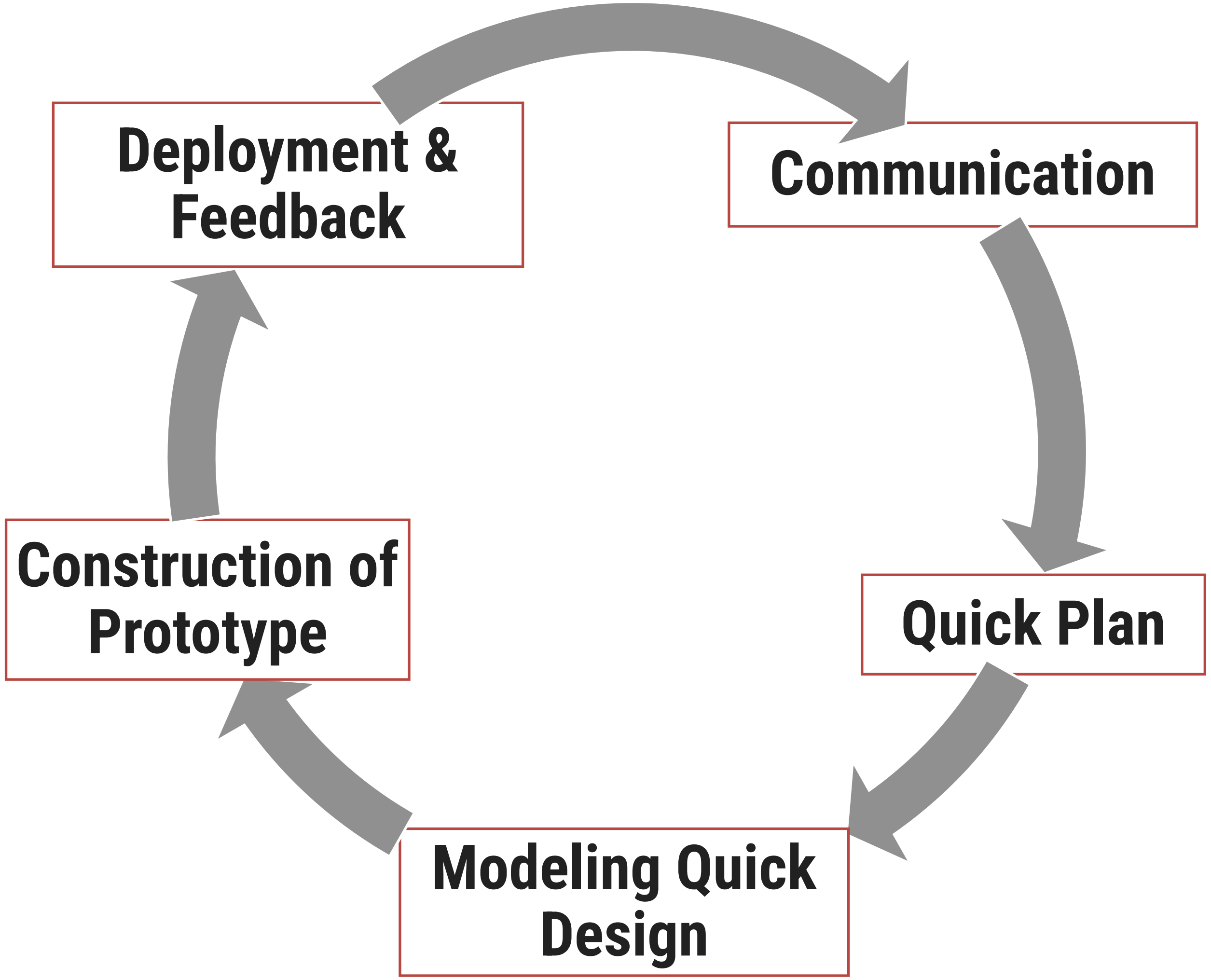
- **Customers have** general **objectives of software** but **do not have detailed requirements** for functions & features.
- **Developers** are **not sure** about **efficiency of an algorithm** & **technical feasibilities**.

---

- It serves as a **mechanism** for **identifying software requirements**.
- Prototype can be serve as “**the first system**”.
- Both stakeholders and software engineers like prototyping model
  - Users get feel for the actual system
  - Developers get to build something immediately

# Prototyping model cont.

## It works as follow



Communication

Quick Plan

Modeling Quick Design

Construction of Prototype

Deployment & Feedback

- **Communicate** with stockholders & **define objective** of Software
- **Identify requirements** & design **quick plan**
- **Model** a quick **design** (focuses on visible part of software)
- **Construct Prototype** & deploy
- Stakeholders **evaluate** this **prototype** and provides **feedback**
- Iteration occurs and **prototype** is **tuned** based on **feedback**

# Prototyping model cont.

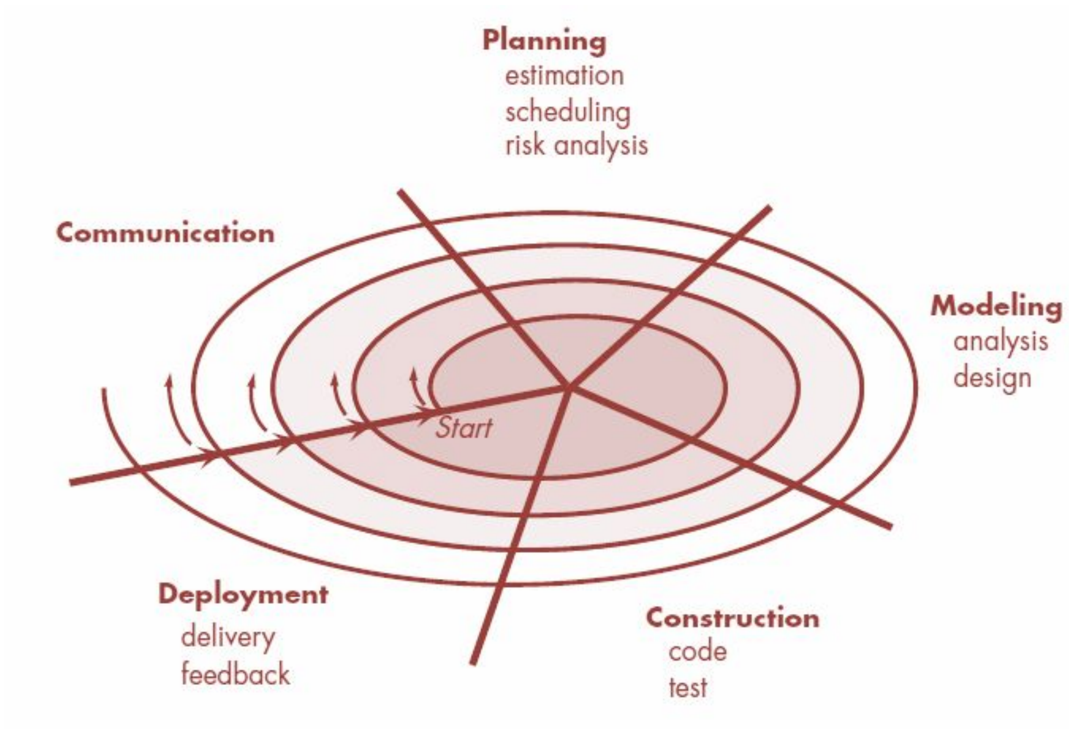
## Problem Areas

- **Customer demand** that “**a few fixes**” be applied to **make** the **prototype a working product**, due to that software quality suffers as a result
- **Developer** often makes **implementation** in order to get a prototype working quickly; **without considering other factors** in mind like OS, Programming language, etc.

## Advantages

- **Users** are actively **involved** in the **development**
- Since in this methodology a working model of the system is provided, the **users get a better understanding** of the **system** being developed
- **Errors** can be **detected** much **earlier**

# The Spiral Model



- It provides the **potential** for **rapid development**.
- Software is developed in a series of evolutionary releases.
- **Early iteration** release might be **prototype** but **later iterations** provides more **complete version of software**.
- It is divided into framework activities (C,P,M,C,D). Each activity represent one segment of the spiral
- **Each pass** through the **planning** region results in **adjustments** to
  - the **project plan**
  - **Cost** & **schedule** based on feedback

# The Spiral Model Cont.

## When to use Spiral Model?

- For development of **large scale / high-risk projects**.
- When costs and **risk evaluation is important**.
- Users are **unsure** of their **needs**.
- **Requirements** are **complex**.
- New product line.
- Significant (**considerable**) **changes** are expected.

## Advantages

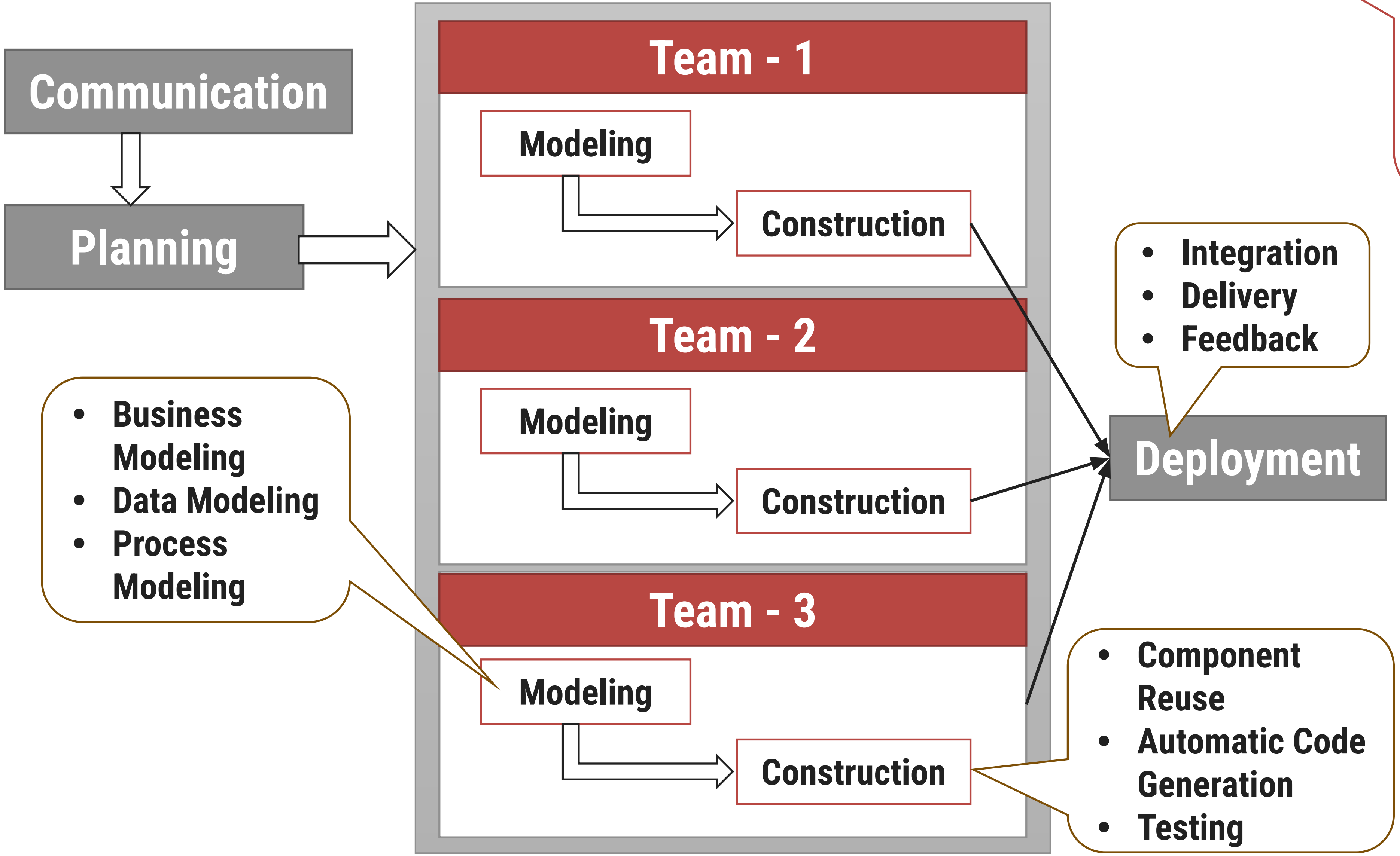
- High amount of risk analysis hence, **avoidance of Risk** is enhanced.
- **Strong approval** and **documentation** control.
- **Additional functionality** can be **added** at a later date.
- **Software** is **produced early** in the Software Life Cycle.

## Disadvantages

- Can be **a costly model** to use.
- Risk analysis **requires highly specific expertise**.
- Project’s success is highly dependent on the risk analysis phase.
- Doesn’t work well for smaller projects.

# Rapid Application Development Model (RAD)

It is a type of **incremental model** in which; **components** or functions are **developed in parallel**.



- Business Modeling
- Data Modeling
- Process Modeling

- Integration
- Delivery
- Feedback

- Component Reuse
- Automatic Code Generation
- Testing

Rapid development is **achieved** by **component based construction**

This can **quickly give** the customer **something to see** and use and to provide feedback.

# Rapid Application Development Model (RAD) Cont.

## Communication

- This phase is used to understand business problem.

## Planning

- Multiple software teams work in parallel on different systems/modules.

## Modeling

- **Business Modeling:** *Information flow* among the business.
  - Ex. What kind of information drives (moves)?
  - Who is going to generate information?
  - From where information comes and goes?
- **Data Modeling:** Information refine into set of *data objects* that are *needed* to support business.
- **Process Modeling:** *Data object* transforms **to** *information flow* necessary to implement business.

## Construction

- It highlighting the *use of pre-existing software component*.

## Deployment

- **Integration of modules** developed by parallel teams, **delivery** of integrated software and **feedback** comes under deployment phase.

# Rapid Application Development Model (RAD) Cont.

## When to Use?

- There is a need to create a **system** that can be **modularized in 2-3 months** of time.
- **High availability** of **designers** and **budget** for modeling along with the cost of automated code generating tools.
- **Resources** with **high** business **knowledge** are available.

## Advantages

- **Reduced** development **time**.
- **Increases reusability** of components.
- **Quick** initial **reviews** occur.
- **Encourages** customer **feedback**.
- Integration from very beginning **solves** a lot **of integration issues**.

## Drawback

- For **large** but scalable **projects**, RAD **requires sufficient human resources**.
- Projects **fail if developers** and **customers** are **not committed** in a much shortened time-frame.
- **Problematic** if system **can not be modularized**.
- **Not appropriate when technical risks are high** (heavy use of new technology).

# Thank You